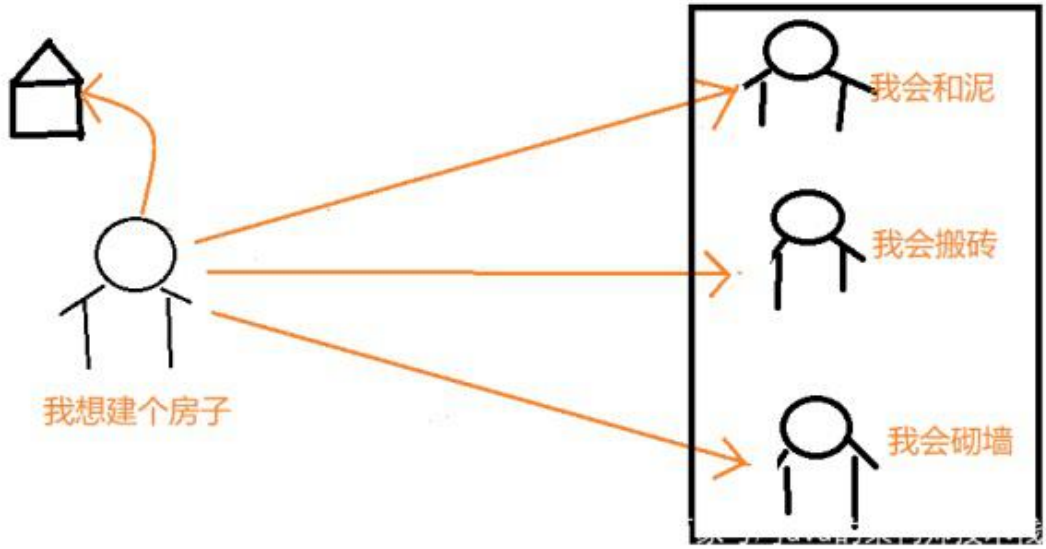
门面模式

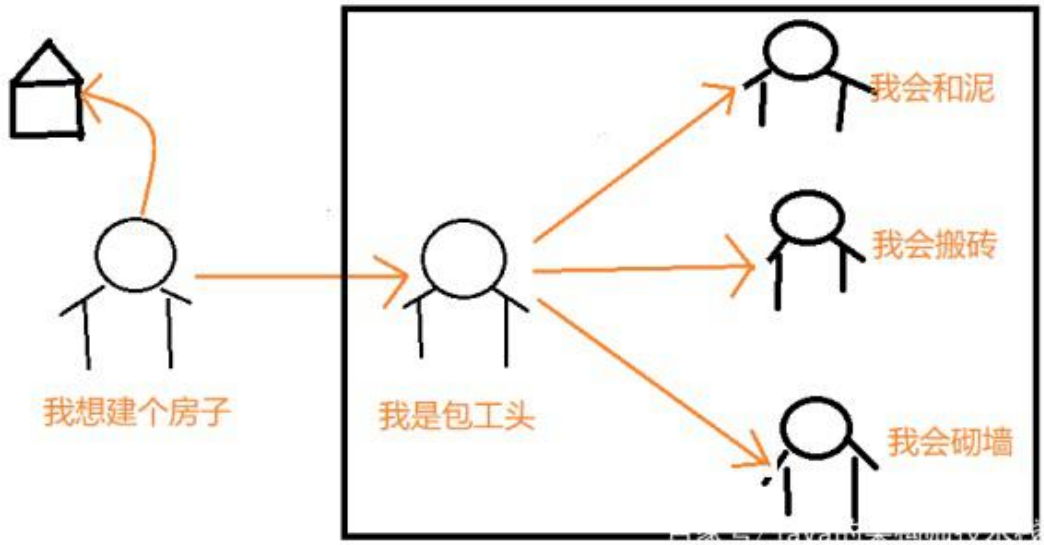
门面模式又叫外观模式。为了深入理解这个模式，首先讲一个例子。这个例子是我参考的网上的例子，叫[老杨叔叔csdn。当然里面只是参考了其例子的思想。

一个例子

有一个人叫张三，在外面奋斗了很多年，终于挣了很多钱，这时候就想着建一栋小洋楼。当然他肯定不会是自己一个人盖房子，所以就想着找其他人。首先他需要找一个会砌墙的、会和泥的、会搬砖的等等。



后来张三一想，这也太麻烦了。我要自己找这些人，还要分别和这些人谈价格，还要给这些人之间调节好关系。要是有一个人给我处理这些事多好！来有人给他推荐了包工头，自己什么都不需要处理，让包工头做这些就好了。张三很开心，立马找了包工头。



现在所有的事都好做了，因为张三只需要跟包工头商量事情，所有的事让包工头去做。

上面的这个例子其实就是门面模式。

门面模式认识

有了例子，那么我们如何使用门面模式去解释呢？在上面的例子当中其实我们发现，整个门面模式分成了两个部分。有三个角色。

第一部分：客户类（client），也就是张三。

第二部分：

门面角色（facade）：在这里指的是包工头。客户端可以调用这个角色的方法。此角色知晓相关的（一个或者多个）子系统的功能和责任。在这里指的是，张三可以调用包工头，包工头也需要知道其手下搬砖、砌墙、和泥等人。子系统角色（subSystem）：在这里指的就是会和泥的、会搬砖的、会砌墙的人。当然，每个子系统都可以被客户端直接调用，或者被门面角色调用。子系统并不知道门面的存在，对于子系统而言，门面仅仅是另外一个客户端而已。**代码实现**

首先是子系统的实现：（三个类）

//搬砖的人public class Person_BanZhuan {public void banzhuan() { System.out.println("我会搬砖。。。。。。"); }}

//和泥的人public class Person_HuoNi {public void huoni() { System.out.println("我会和泥。。。。。。"); }}

//砌墙的人public class Person_QiQiang {public void qiqiang() { System.out.println("我会砌墙。。。。。。"); }}

接下来是门面角色（facade）：包工头

//包工头public class BaoGongTou {//满足客户端张三需要的功能,建房子 public void build(){ Person_BanZhuan a = new Person_BanZhuan(); a.banzhuan(); Person_HuoNi b = new Person_HuoNi(); b.huoni(); Person_QiQiang c = new Person_QiQiang(); c.qiqiang(); }}

最后呢就是客户端类：张三

//我是张三public class Client {public static void main(String[] args) { BaoGongTou facade=new BaoGongTou(); facade.build(); }}

门店模式特点：

1、子系统可以有选择的暴露方法

门面模式还有一个附带的好处，就是能够有选择性地暴露方法。一个模块中定义的方法可以分成两部分，一部分是给子系统外部使用的，一部分是子系统内部模块之间相互调用时使用的。有了Facade类，那么用于子系统内部模块之间相互调用的方法就不用暴露给子系统外部了。

2、一个门面模式可以有很多个门面类

在门面模式中，通常只需要一个门面类。如果一个系统有好几个子系统的话，每一个子系统都有一个门面类，整个系统可以有数个门面类。

3、不能为子系统增加新行为

门面模式的用意是为子系统提供一个集中化和简化的沟通管道，而不能向子系统加入新的行为。就比如包工头的作用只是调度其他人工作的，但是自己不工作。

4、松耦合、而且使用简单。

用户与子系统解耦，屏蔽子系统；可以提高子系统的独立性；并且客户类不需要知道子系统的内部构造。

与代理模式的区别：

这个模式乍一看还真的很像代理模式，其实还是有很大的区别的。

比如说他们都引入了中介，起到了代理的功能。但是代理模式只代理一个类，而且代理类与原类实现相同的抽象。门面类就不一样了，他代理的是一系列类，与子系统可以有不同的抽象。